МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ   
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
имени И.С. ТУРГЕНЕВА»

ОТЧЕТ

О РазработкЕ Электронного сервиса по социально-психолого-педагогическому сопровождению

семей с детьми раннего возраста

Орёл 2021

Содержание

[ВВЕДЕНИЕ 8](#_Toc75973633)

[1. Обзор инструментов для создания Электронного сервиса 11](#_Toc75973634)

[1.1 Сведения о Next.js, React и не обходимых истнументах 11](#_Toc75973635)

[1.2 Рендеринг на стороне сервера 16](#_Toc75973636)

[1.3 Дополнительные инструменты и библиотеки для полноценного сервиса, способного поддерживаться и обновляться 19](#_Toc75973637)

[2 РАЗРАБОТКА САЙТА 27](#_Toc75973638)

[2.1 Создание базового шаблона для сайта 2](#_Toc75973639)7

[2.2 Разработка чат виджета на основе Socket.io 3](#_Toc75973640)7

[3 ФУНКЦИОНАЛ САЙТА 47](#_Toc75973641)

[3.1 Виджет чата 4](#_Toc75973642)7

[3.2 Форум с оценками и возможностью менторства 53](#_Toc75973643)

[3.3 Панель администратора для контроля контента управления сайтом 6](#_Toc75973644)0

[ЗАКЛЮЧЕНИЕ 71](#_Toc75973645)

[СПИСОК ИСТОЧНИКОВ 7](#_Toc75973646)2

# ВВЕДЕНИЕ

Государственная программа Российской Федерации «Развитие образования» на 2019-2025 годы предусматривает федеральный проект «Поддержка семей, имеющих детей». Целью проекта является создание условий для повышения компетентности родителей обучающихся в вопросах образования и воспитания, в том числе для раннего развития детей в возрасте до трех лет, путем предоставления в 2024 году не менее 20 млн. услуг психолого-педагогической, методической и консультативной помощи родителям (законным представителям) детей, а также гражданам, желающим принять на воспитание в свои семьи детей, оставшихся без попечения родителей.

В настоящее время в Российской Федерации, в том числе в Орловской области, в основном обеспечена 100% доступность дошкольного образования для детей в возрасте от 3 до 7 лет. Однако проблема охвата образовательными услугами детей раннего возраста (до 3-х лет) сохраняет свою социальную значимость. Особый ракурс рассмотрения данная проблема приобретает в свете основных положений, сформулированных в Концепции демографической политики Российской Федерации на период до 2025 года. В частности, предложенный в документе комплекс мер должен обеспечить увеличение численности населения России к 2025 году до 145 млн. человек. Прежде всего, демографический рост ожидается за счет увеличения рождаемости, что приведет к увеличению числа семей, имеющих детей раннего возраста. Актуализирует данную проблему пересмотр возраста трудоспособности населения. Увеличение пенсионного возраста снижает возможности привлечения представителей третьего поколения к воспитанию внуков, что осложняет уход за детьми раннего возраста в семье. В этом семье могут помочь специалисты педагоги-психологи.

Несомненно, каждый педагог-психолог должен проходить курсы повышения квалификации и переподготовки для того, чтобы оставаться востребованным специалистом. А так как в наши дни набирают популярность онлайн-курсы, то именно они могут помочь педагогам-психологам в удобной и комфортной для них обстановке пройти данный курс. Стоит отметить то, что по его окончании специалисту выдается сертификат.

**Степень разработанности проблемы**

На данный момент не удалось обнаружить онлайн-сервисы в области психолого-педагогического сопровождения развития детей раннего возраста. Это обуславливает необходимость разработки такого электронного ресурса.

**Предмет исследования**

Разработка электронного ресурса для поддержки семей с детьми раннего возраста.

**Объект исследования**

Система поддержки пользователей, в качестве которых выступают семьи с детьми раннего возраста.

**Цель работы**

Создать сайт, который позволит пользователям просматривать обучающие материалы, оставлять коментарии, задавать вопросы специалистам.

**Основные задачи исследования**

Для достижения поставленной цели необходимо решить следующие задачи:

1. Выбрать технологии для создания онлайн-ресурса.
2. Разработать шаблон сайта.
3. Оснастить сайт необходимым функционалом, таким как чат и форум, на котором пользователи смогут следить за постами и их участниками.

**Методы исследования**

При исследовании применялся теоретический анализ литературы и материалов сети Интернет, методы структурного и объектно-ориентированного программирования.

**Теоретическое и практическое значение работы**

Работа имеет теоретическое и практическое значение, так какразработанный ресурс можно активно использовать среди семей с детьми раннего возраста.

**Основные инструменты**

Основными инструментами для разработки в данной выпускной квалификационной работе является фреймворкиNext.js, React.js, WebSocket, MongoDB

**Структура работы**

Работа состоит из введения, трёх глав, заключения и списка источников.

Во введении рассматривается актуальность работы, ставится цель и обозначаются задачи, необходимые для достижения поставленной цели.

Первая глава представляет собой совокупность теоретических данных об основных подходах к созданию подобных ресурсов.

Во второй главе описывается создание ключевых компонетов сайта.

В третьей главе описывается логика работы сайта, основного функционала.

В заключении делаются выводы по проделанной работе и рассматриваются дальнейшие перспективы применения разработанного программного обеспечения.

В конце работы приводится список использованных источников.

# 1 Обзор инструментов для создания Электронного сервиса

## **1.1 Сведения о Next.js, Reactи необходимых инструментах**

Next.js– это фреймворк с открытым исходным кодом, который упрощает процесс разработки на React, абстрагируя общие и избыточные задачи (например, маршрутизацию) в относительно более простые и мощные API. Таким образом, можно сосредоточиться на написании своих приложений, а не изобретать колесо.

Next.js предназначен для статических сайтов и был хорошо спроектирован для этой цели. Но он также продвигается дальше с помощью инкрементальной статической регенерации, которая хорошо сочетается с существующими функциями, чтобы сделать разработку удобнее.

Изначльно данная среда была разработана компанией Vercel, которая занимается разработкой инструментов для разработчиков, а в последствии улучшалась тысячамии других разработчиков по всему миру. Она обеспечивает такие функции, как рендеринг на стороне сервера и создание статических веб-сайтов для веб-приложений на основе React.Традиционные приложения на React отображают весь свой контент в браузере на стороне клиента, Next.js используется для расширения этой функциональности, включая приложения, отображаемые на стороне сервера. Конкретно рендеринг на стороне сервера решает проблему долгой первой подгрузки всего необходимого кода и положительно влияет на иднексацию поисковыми роботами.

В последнее время разработчики Next.js утверждают, что их продукт способен сразу производить готовый к боевому использованию проект, и, дейсвительно, фреймворк предлагает множество функций, которые помогут вывести веб-сайт React с нуля на уровень использования большой аудиторией.

React– это библиотека JavaScript, созданная для создания быстрых и интерактивных пользовательских интерфейсов для веб-приложений и мобильных приложений. Это компонентная интерфейсная библиотека с открытым исходным кодом, отвечающая только за уровень представления приложения. В архитектуре ModelViewController (MVC) уровень представления отвечает за то, как приложение выглядит и ощущается. React был создан Джорданом Уолком, инженером-программистом в Facebook.

Популярность React сегодня превзошла популярность всех других интерфейсных сред разработки по ряду причин.

Простое создание динамических приложений: React упрощает создание динамических веб-приложений, поскольку требует меньше кода и предлагает больше функциональных возможностей, в отличие от JavaScript, где разработка часто очень быстро усложняется.

Повышенная производительность: React использует Virtual DOM, что ускоряет создание веб-приложений. Виртуальная модель DOM сравнивает предыдущие состояния компонентов и обновляет только те элементы в реальной модели DOM, которые были изменены, вместо повторного обновления всех компонентов, как это делают обычные веб-приложения.

Компоненты многократного использования: компоненты являются строительными блоками любого приложения React, а одно приложение обычно состоит из нескольких компонентов. Эти компоненты имеют свою логику и элементы управления, и их можно повторно использовать во всем приложении, что, в свою очередь, значительно сокращает время разработки приложения.

Однонаправленный поток данных: React следует за однонаправленным потоком данных. Это означает, что при разработке приложения React разработчики часто вкладывают дочерние компоненты в родительские компоненты. Поскольку данные передаются в одном направлении, становится проще отлаживать ошибки и знать, где в приложении возникает проблема в данный момент.

Небольшая кривая обучения: React легко освоить, поскольку он в основном сочетает в себе базовые концепции HTML и JavaScript с некоторыми полезными дополнениями. Тем не менее, как и в случае с другими инструментами и фреймворками, вам нужно потратить некоторое время, чтобы получить правильное представление о библиотеке React.

Его можно использовать для разработки как веб-приложений, так и мобильных: мы уже знаем, что React используется для разработки веб-приложений, но это еще не все, что он может. Существует фреймворк под названием React Native, производный от самого React, который очень популярен и используется для создания красивых мобильных приложений. Итак, на самом деле React можно использовать для создания как веб-приложений, так и мобильных.

Специальные инструменты для легкой отладки: Facebook выпустил расширение Chrome, которое можно использовать для отладки приложений React. Это ускоряет и упрощает процесс отладки веб-приложений React.

Вышеупомянутые причины более чем оправдывают популярность библиотеки React и то, почему она принимается большим количеством организаций и предприятий. Теперь давайте познакомимся с функциями React.

JSX - расширение синтаксиса JavaScript. Он используется с React для описания того, как должен выглядеть пользовательский интерфейс. Используя JSX, мы можем писать HTML-структуры в том же файле, который содержит код JavaScript. Это упрощает понимание и отладку кода, поскольку позволяет избежать использования сложных структур DOM JavaScript.

React сохраняет в памяти облегченное представление «реальной» модели DOM, известное как «виртуальная» модель DOM (VDOM). Манипулирование реальной DOM намного медленнее, чем управление VDOM, потому что на экране ничего не отображается. Когда состояние объекта изменяется, VDOM изменяет только этот объект в реальной модели DOM, а не обновляет все объекты.

На данный момент все это может показаться немного сложным, поэтому давайте сначала разберемся, что такое DOM, а затем рассмотрим, как VDOM и реальный DOM взаимодействуют друг с другом.

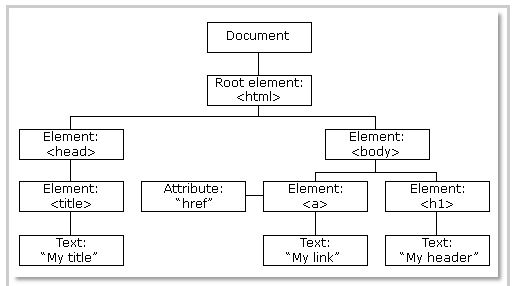


Рисунок 1 – Объектная модель документа

Когда состояние объекта изменяется в приложении React, VDOM обновляется. Затем он сравнивает свое предыдущее состояние и обновляет только те объекты в реальной DOM вместо обновления всех объектов. Это ускоряет процесс, особенно по сравнению с другими интерфейсными технологиями, которые должны обновлять каждый объект, даже если в веб-приложении изменяется только один объект.

React использует VDOM, благодаря чему веб-приложения работают намного быстрее, чем приложения, разработанные с использованием альтернативных интерфейсных фреймворков. React разбивает сложный пользовательский интерфейс на отдельные компоненты, позволяя нескольким пользователям работать над каждым компонентом одновременно, тем самым ускоряя время разработки.

React выходит за рамки простого дизайна пользовательского интерфейса и имеет множество расширений, которые предлагают полную поддержку архитектуры приложения. Он обеспечивает рендеринг на стороне сервера, что влечет за собой рендеринг обычно только клиентского веб-приложения на сервере, а затем отправляет клиенту полностью отрисованную страницу. Он также широко использует Flux и Redux при разработке веб-приложений. Наконец, есть ReactNative, популярный фреймворк, производный от React, используемый для создания кросс-совместимых мобильных приложений.

Односторонняя привязка данных в React делает все модульным и быстрым. Однонаправленный поток данных означает, что, когда разработчик разрабатывает приложение React, он часто вкладывает дочерние компоненты в родительские компоненты. Таким образом, разработчик знает, где и когда возникает ошибка, что дает ему лучший контроль над всем веб-приложением.

Приложения React легко тестировать благодаря большому сообществу разработчиков. Facebook даже предоставляет небольшое расширение для браузера, которое ускоряет и упрощает отладку React.

Компоненты являются строительными блоками любого приложения React, и одно приложение обычно состоит из нескольких компонентов. Компонент по сути — это часть пользовательского интерфейса. React разбивает пользовательский интерфейс на независимые, многоразовые части, которые можно обрабатывать отдельно.

**Функциональные компоненты**: эти компоненты не имеют собственного состояния и содержат только метод рендеринга, поэтому их также называют компонентами без состояния. Они могут получать данные из других компонентов в качестве свойств.

**Компоненты класса**: эти компоненты могут удерживать свое состояние и управлять им, а также имеют отдельный метод рендеринга для возврата JSX на экран. Их также называют компонентами с отслеживанием состояния, поскольку они могут иметь состояние.

**Состояние** — это встроенный объект React, который используется для хранения данных или информации о компоненте. Состояние компонента может меняться со временем; всякий раз, когда он изменяется, компонент повторно отрисовывается. Изменение состояния может происходить в ответ на действия пользователя или события, генерируемые системой, и эти изменения определяют поведение компонента и то, как он будет отображаться.

**Props**– это аббревиатура от **properties**. Это встроенный объект React, который хранит значение атрибутов тега и работает аналогично атрибутам HTML. Он обеспечивает способ передачи данных от одного компонента к другим компонентам так же, как аргументы передаются в функции.

Таким образом, компоненты, состояние и свойства – это фундаментальные концепции React. Прежде чем начинать работать с React, есть несколько концепций, с которыми нужно быть знакомым.

## **1.2. Рендеринг на стороне сервера**

Рендеринг на стороне сервера (SSR) — это процесс рендеринга веб-страниц на сервере и их передачи в браузер (на стороне клиента) вместо их рендеринга в браузере. SSR отправляет клиенту полностью отрисованную страницу; пакет клиентского JavaScript берет на себя и позволяет фреймворку SPA работать.

Это означает, что, еслиприложение отображается на стороне сервера, контент извлекается с сервера и передается в браузер для отображения пользователю. Рендеринг на стороне клиента отличается: пользователю нужно будет перейти на страницу до того, как браузер получит данные с сервера, а это означает, что пользователю придется подождать несколько секунд, прежде чем браузер получит контент для этой страницы. Приложения с включенным SSR называются приложениями, отображаемыми на стороне сервера.

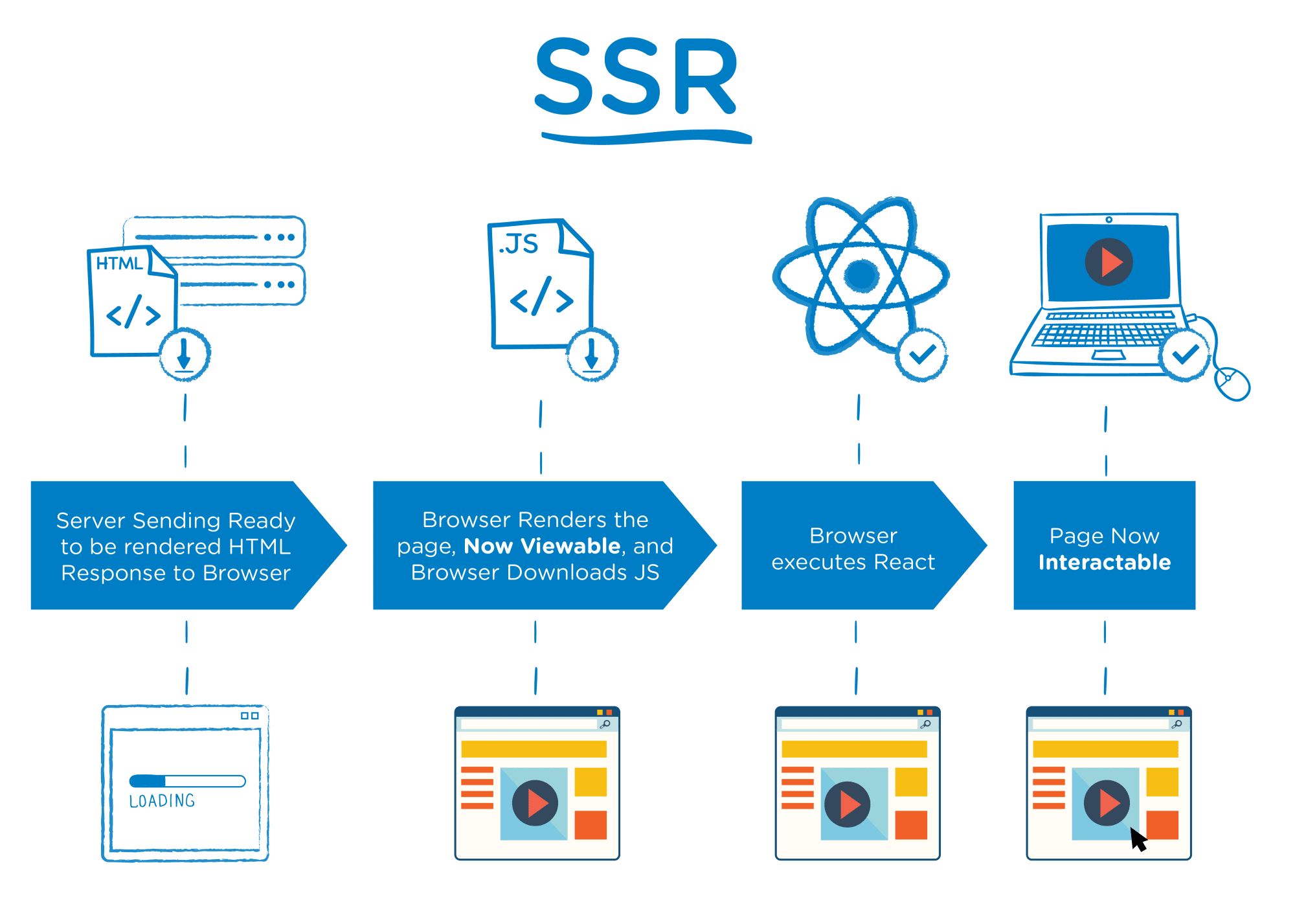


Рисунок 2 – Как работает рендеринг на стороне сервера

Этот подход хорош, если вы создаете сложное приложение, которое требует взаимодействия с пользователем, полагается на базу данных или содержимое, которого часто меняется. Если контент меняется часто, пользователям нужно будет сразу увидеть обновления. Этот подход также хорош для приложений, которые адаптируют контент в зависимости от того, кто его просматривает, и хранят пользовательские данные, такие как адреса электронной почты и пользовательские предпочтения, а также занимаются поисковой оптимизацией. Примером может служить крупная платформа электронной коммерции или социальных сетей. Давайте посмотрим на некоторые преимущества SSR для ваших приложений.

Так же следует решить, или чтобы HTML-файлы уже были сгенерированы (статический рендеринг), или необходимо сначала обработать запрос, а затем создать (соответственно) HTML-файл на лету.

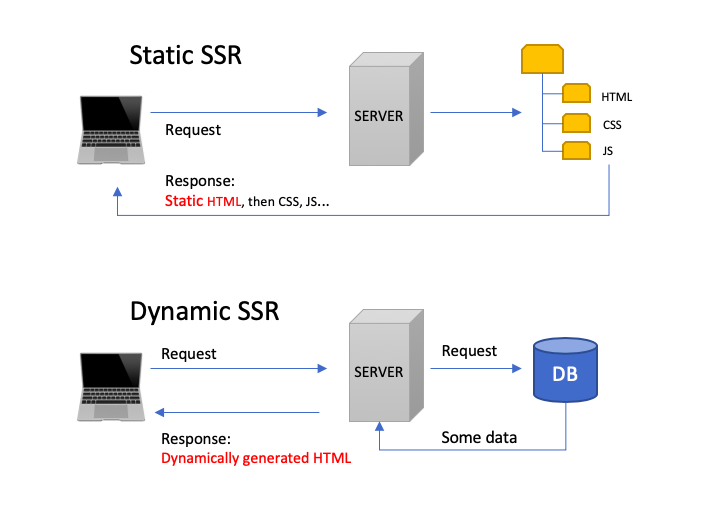


Рисунок 3 – Статический и динамический рендеринг

**Статический SSR**

* Лучшая производительность: серверу не нужно создавать файлы (они уже есть);
* CDN/Caching/Gzip: благодаря этому можно сжимать и кэшировать файл рядом с пользователями, чтобы снизить задержку при сетевом вызове;
* Требуется «тяжеловесная» CSR: поскольку HTML-файл будет довольно простым, клиент будет отвечать за выборку и рендеринг данных;
* Требуется управлять процессом генерации html-файла: вы можете просто написать html-файл вручную, но вы, скорее всего, закончите с каким-то менеджером процессов (например, gulp), который может вызывать какой-то процесс для генерации html. Это процесс только для разработчиков, вы можете сделать это на своем компьютере до развертывания;
* Вам по-прежнему необходимо реализовать ApiServer, потому что это правда, что вы можете обслуживать файлы html, css и js как статические файлы, но данные json, вероятно, всегда будут динамическими и зависимыми от пользователя;

**Динамический SSR**

* Не самая лучшая производительность: серверу необходимо генерировать ответ на каждый запрос, поэтому производительность может быть хорошей, но не лучшей;
* Кеширование можно сделать, но это сложнее;
* Вам не нужен процесс сборки для hmtl-файлов (вы создаете html-ответ на лету);
* Нет необходимости внедрять отдельный сервер для обработки требований к данным («ApiServer»);
* Клиентская сторона может быть легкой (но, возможно, она будет тяжелой по другим причинам);

## 

## **1.3 Дополнительные инструменты и библиотеки для полноценного сервиса, способного легко поддерживаться и обновляться**

NGINX — это программное обеспечение с открытым исходным кодом для веб-обслуживания, обратного проксирования, кэширования, балансировки нагрузки, потоковой передачи мультимедиа и многого другого. Он начинался как веб-сервер, рассчитанный на максимальную производительность и стабильность. Помимо возможностей HTTP-сервера, NGINX также может функционировать как прокси-сервер для электронной почты (IMAP, POP3 и SMTP), а также как обратный прокси-сервер и балансировщик нагрузки для серверов HTTP, TCP и UDP.

Целью NGINX было создание самого быстрого веб-сервера, и поддержание этого превосходства по-прежнему является центральной целью проекта. NGINX неизменно превосходит Apache и другие серверы в тестах, измеряющих производительность веб-серверов. Однако с момента первоначального выпуска NGINX веб-сайты расширились от простых HTML-страниц до динамического многогранного контента. NGINX вырос вместе с ним и теперь поддерживает все компоненты современного Интернета, включая WebSocket, HTTP / 2, gRPC и потоковую передачу нескольких видеоформатов (HDS, HLS, RTMP и другие).

Хотя NGINX прославился как самый быстрый веб-сервер, масштабируемая базовая архитектура оказалась идеальной для многих веб-задач, помимо обслуживания контента. Поскольку он может обрабатывать большой объем подключений, NGINX обычно используется в качестве обратного прокси-сервера и балансировщика нагрузки для управления входящим трафиком и его распределения на более медленные вышестоящие серверы – от устаревших серверов баз данных до микросервисов.

NGINX также часто размещается между клиентами и вторым веб-сервером в качестве терминатора SSL / TLS или веб-ускорителя. Выступая в качестве посредника, NGINX эффективно обрабатывает задачи, которые могут замедлить работу вашего веб-сервера, например согласование SSL / TLS или сжатие и кэширование контента для повышения производительности. Динамические сайты, созданные с использованием чего угодно, от Node.js до PHP, обычно развертывают NGINX в качестве кэша контента и обратного прокси-сервера, чтобы снизить нагрузку на серверы приложений и максимально эффективно использовать базовое оборудование.

Docker — это открытая платформа для разработки, доставки и запуска приложений. Docker позволяет отделить приложения от инфраструктуры, чтобы вы могли быстро доставлять программное обеспечение. С помощью Docker вы можете управлять своей инфраструктурой так же, как вы управляете своими приложениями. Воспользовавшись методологиями Docker для быстрой доставки, тестирования и развертывания кода, вы можете значительно сократить задержку между написанием кода и его запуском в производственной среде.

Docker предоставляет возможность упаковывать и запускать приложение в слабо изолированной среде, называемой контейнером. Изоляция и безопасность позволяют запускать множество контейнеров одновременно на одном хосте. Контейнеры легкие и содержат все необходимое для запуска приложения, поэтому вам не нужно полагаться на то, что в настоящее время установлено на хосте. Вы можете легко обмениваться контейнерами во время работы и быть уверенным, что все, с кем вы делитесь, получают один и тот же контейнер, который работает одинаково.

Платформа Docker на основе контейнеров обеспечивает высокую переносимость рабочих нагрузок. Контейнеры Docker могут работать на локальном ноутбуке разработчика, на физических или виртуальных машинах в центре обработки данных, у облачных провайдеров или в нескольких средах. Портативность и легкость Docker также позволяют легко динамически управлять рабочими нагрузками, увеличивая или уменьшая масштабы приложений и сервисов в соответствии с потребностями бизнеса, практически в реальном времени. Выполнение дополнительных рабочих нагрузок на том же оборудовании. Докер легкий и быстрый. Он представляет собой жизнеспособную и экономичную альтернативу виртуальным машинам на основе гипервизора, поэтому вы можете использовать больше своих вычислительных мощностей для достижения своих бизнес-целей. Docker идеально подходит для сред с высокой плотностью, а также для малых и средних развертываний, когда вам нужно делать больше с меньшими ресурсами.

Docker использует архитектуру клиент-сервер. Клиент Docker общается с демоном Docker, который выполняет тяжелую работу по созданию, запуску и распространению ваших контейнеров Docker. Клиент и демон Docker могут работать в одной системе, или вы можете подключить клиент Docker к удаленному демону Docker. Клиент Docker и демон обмениваются данными с помощью REST API, через сокеты UNIX или сетевой интерфейс. Другой клиент Docker — это Docker Compose, который позволяет работать с приложениями, состоящими из набора контейнеров.

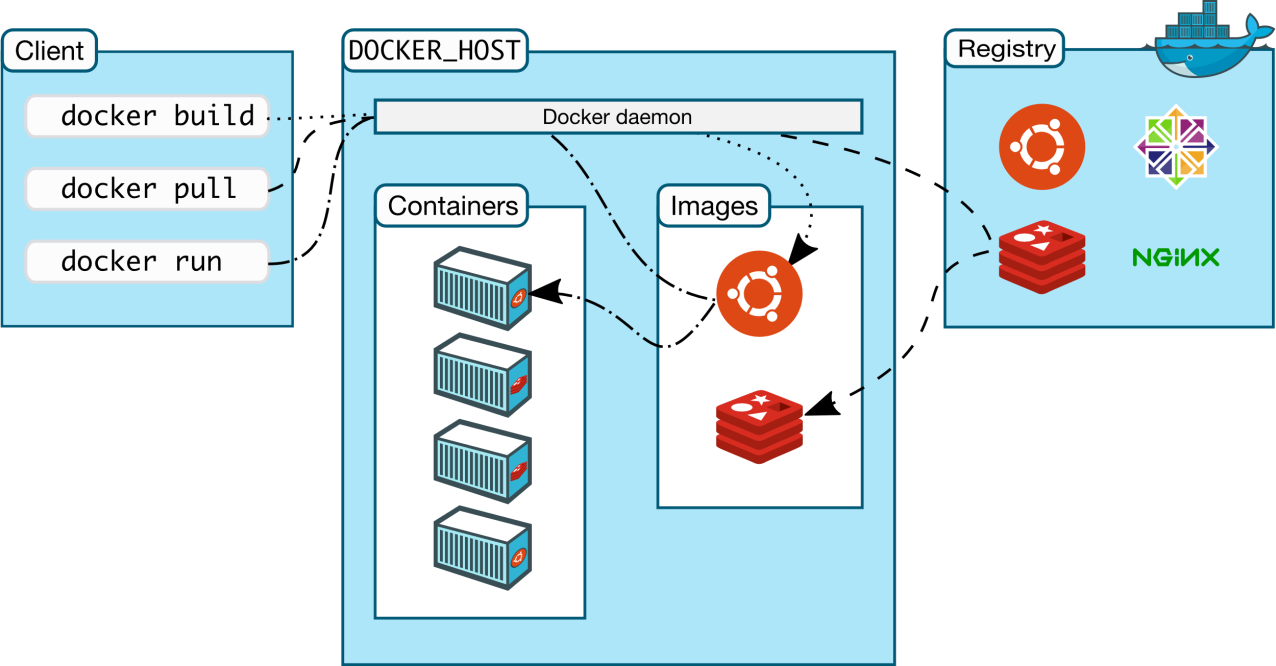


Рисунок 4 – Схема общей архитектуры Docker

Часто у разработчиков при работе с Dockerвозникает проблема понимания что такое образ, однако это просто шаблон только для чтения с инструкциями по созданию контейнера Docker. Часто изображение основано на другом изображении с некоторыми дополнительными настройками. Например, вы можете создать образ, основанный на образе ubuntu, но установив веб-сервер Apache и ваше приложение, а также детали конфигурации, необходимые для запуска вашего приложения.

Вы можете создавать свои собственные образы или использовать только те, которые созданы другими и опубликованы в реестре. Чтобы создать свой собственный образ, вы создаете Dockerfile с простым синтаксисом для определения шагов, необходимых для создания образа и его запуска. Каждая инструкция в Dockerfile создает слой в изображении. Когда вы меняете Dockerfile и перестраиваете образ, перестраиваются только те слои, которые были изменены. Это часть того, что делает образы такими легкими, маленькими и быстрыми по сравнению с другими технологиями виртуализации.

Контейнер — это исполняемый экземпляр изображения. Вы можете создавать, запускать, останавливать, перемещать или удалять контейнер с помощью Docker API или интерфейса командной строки. Вы можете подключить контейнер к одной или нескольким сетям, присоединить к нему хранилище или даже создать новый образ на основе его текущего состояния.

По умолчанию контейнер относительно хорошо изолирован от других контейнеров и его хост-машины. Вы можете контролировать, насколько изолирована сеть, хранилище или другие базовые подсистемы контейнера от других контейнеров или от хост-машины.

Контейнер определяется его образом, а также любыми параметрами конфигурации, которые вы ему предоставляете при создании или запуске. Когда контейнер удаляется, любые изменения его состояния, которые не хранятся в постоянном хранилище, исчезают.

**WebSocket** - это протокол интернет-связи с соответствующей интересной функцией: он обеспечивает полнодуплексный канал через одно TCP-соединение.

С помощью WebSockets клиент и сервер могут общаться друг с другом в режиме реального времени, как будто они участвуют в телефонном звонке: после подключения клиент может получать данные с сервера без необходимости постоянно обновлять страница в Интернете. С другой стороны, сервер также сможет получать данные в реальном времени от клиента внутри того же соединения.

Кроме того, WebSockets управляется событиями: и сервер, и клиент могут реагировать на события и сообщения. Мы можем прослушивать событие подключения, запускать функцию, когда новый пользователь подключается к серверу, отправлять сообщение (в основном событие) через сокет и многое другое.

WebSockets открыл целый мир возможностей для веб-разработчиков. Если вам интересно, как реализовать эту фантастическую технологию в ваших приложениях Node.js, то ответ –Socket.IO, один из самых популярных движков реального времени для Node.js.

Следует иметь в виду, что Socket.IO не является реализацией WebSocket. Авторы заявляют, что «Socket.IO действительно использует WebSocket в качестве транспорта, когда это возможно, но клиент WebSocket не сможет подключиться к серверу Socket.IO, а клиент Socket.IO не сможет подключиться к серверу WebSocket.».

Протокол WebSocket позволяет осуществлять двустороннюю связь между клиентом и сервером. Этот процесс аналогичен тому, как совершаются звонки на вашем телефоне: сначала вы устанавливаете соединение, а затем вы можете начать общаться друг с другом. Протокол WebSocket используется практически везде – от многопользовательских браузерных игр до приложений чата.

SSL расшифровывается как Secure Sockets Layer и, короче говоря, это стандартная технология для обеспечения безопасности интернет-соединения и защиты любых конфиденциальных данных, которые пересылаются между двумя системами, не позволяя злоумышленникам читать и изменять любую передаваемую информацию, включая потенциальные личные данные. Две системы могут быть сервером и клиентом (например, веб-сайт покупок и браузер) или сервером с сервером (например, приложение с личной идентифицируемой информацией или с информацией о заработной плате).

Это достигается за счет того, что любые данные, передаваемые между пользователями и сайтами или между двумя системами, остаются невозможными для чтения. Он использует алгоритмы шифрования для шифрования данных при передаче, не позволяя хакерам прочитать их, когда они передаются по соединению. Эта информация может быть конфиденциальной или личной, включая номера кредитных карт и другую финансовую информацию, имена и адреса.

TLS (Transport Layer Security) – это просто обновленная, более безопасная версия SSL. Мы по-прежнему называем наши сертификаты безопасности SSL, потому что это более часто используемый термин, но когда вы покупаете SSL у DigiCert, вы фактически покупаете самые последние сертификаты TLS с возможностью шифрования ECC, RSA или DSA.

HTTPS (защищенный протокол передачи гипертекста) отображается в URL-адресе, когда веб-сайт защищен сертификатом SSL. Подробную информацию о сертификате, включая орган, выдавший сертификат, и название компании владельца веб-сайта, можно просмотреть, щелкнув значок замка на панели браузера.

Протокол WebSocket двунаправленный, полнодуплексный, что означает, что он может одновременно и получать, и передавать информацию. Веб-сокет делает это множество раз в одном открытом соединении. У такого соединения и скорость выше, чем у HTTP.

У веб-сокетов также есть возможность шифровать передаваемые данные, для этого используется надстройка над протоколом — WSS. Если передаваемые данные не зашифрованы, они становятся объектом для привлечения таких угроз, как несанкционированный доступ к клиенту третьих сторон, использование вредоносного ПО. Специальные надстройки протоколов передачи данных кодируют информацию на стороне отправителя и раскодируют на стороне получателя, оставляя ее зашифрованной для любых посредников. Так достигается безопасный транспортный уровень.

Для наглядности покажем, в чем различия технологий HTTP и WebSocket, на примере их работы в чатах.Чтобы пользователи могли получать новые сообщения, браузер периодически опрашивал сервер: есть ли новые сообщения для пользователя? Каждый такой запрос устанавливал новое соединение и создавал лишнюю сетевую нагрузку. Веб-сокеты позволяют установить одно соединение, а сервер сам отправит новые сообщения, когда они появятся.

# 2 РАЗРАБОТКА САЙТА

## **2.1 Создание базового шаблона для сайта**

Чтобы загрузить новое приложение Next.js, нам нужно создать новый каталог проекта и установить необходимые зависимости с помощью NPM (Node.js Package Manager).

$ mkdir your-project

$ cd your-project

Затем запустите эту команду, чтобы запустить новое приложение Node.js и создать файл package.json в корне вашего проекта:

$ npminit -y

Команда npm init -y создает файл package.json в корне каталога вашего проекта. Флаг -y инициализирует файл со значениями по умолчанию. Файл package.json позволит вам легко установить и использовать зависимости пакетов NPM в вашем проекте. Это также упростит такие вещи, как совместное использование вашего проекта с другими разработчиками, если вы захотите сделать это в будущем. Теперь, когда у нас есть файл package.json, мы можем установить необходимые зависимости пакета NPM для вашего веб-сайта Next.js. Для начала нам понадобятся NPM-пакеты Next, React и React-Dom. Можно установить их все сразу с помощью этой команды:

$npminstall --savenextreactreact-dom

Когда они завершат установку, вы заметите, что в вашем проекте был создан новый каталог node\_modules. В этом каталоге хранятся все установленные зависимости для вашего проекта. Если вы заглянете внутрь, то заметите, что там находятся три пакета NPM, которые вы установили, и все их подчиненные зависимости.

Поскольку мы использовали флаг --save в вашей команде установки npm, три зависимости будут перечислены в разделе «dependencies» вашего файла package.json. В будущем, когда вы поделитесь своим кодом с другими, все пакеты в этом списке будут установлены при первоначальной настройке приложения или при запуске команды npm install. Теперь, когда у нас установлены ваши зависимости, нам нужен способ запустить ваше приложение.

Вфайлеpackage.json должен быть следующий скрипт:

"scripts": {

"dev": "next"

},

Скрипт «dev» используется для запуска приложения, когда вы находитесь в режиме разработки. Это означает, что ваш код будет работать со специальной обработкой ошибок, горячей перезагрузкой и другими функциями, которые сделают процесс разработки более приятным.Позже мы добавим в этот раздел дополнительные скрипты для обработки рабочих версий вашего приложения.В вашем терминале запустите приложение в режиме разработки с помощью этой команды:

$ npm run dev

Каталог / pages будет содержать все страницы вашего веб-сайта, а файл index.js будет служить вашей домашней страницей по пути URL-адреса "/". Имя каждого файла в каталоге / pages будет соответствовать пути URL-адреса в браузере при посещении вашего веб-сайта. Например, файл с путем /pages/articles.js будет иметь URL-адрес, который отображается в браузере как «/ article». Все это автоматически выполняет Next.js. Файл /pages/index.js является единственным исключением, поскольку он служит домашней страницей по пути «/». Нам нужно добавить код в ваш файл /pages/index.js, чтобы дать Next.js что-нибудь для рендеринга.

Общий вид файла /pages/index.js должен быть таким:

import React, { Component } from "react"

export default class extends Component {

render () {

return (

<div>Your Next.js App</div>

)

}

}

Одна из замечательных особенностей Next.js — это функции рендеринга на стороне сервера, которые он предоставляет. Вы можете извлекать и получать данные из внешнего API до того, как ваша веб-страница отобразится. Чтобы продемонстрировать это, мы будем использовать бесплатный общедоступный API под названием RandomDog. Каждый раз, когда делается запрос к их API, возвращается случайное изображение собаки. Используя этот API, мы обновим страницу Next.js, которую вы только что создали, чтобы отображать случайное изображение собаки при каждой загрузке страницы. Мы будем делать запросы с помощью пакета NPM под названием Isomorphic-Unfetch. Этот пакет отлично подходит для Next.js, потому что он работает как в клиентской, так и в серверной среде. Установите пакет NPM в свой проект с помощью этой команды:

$ npminstall --saveisomorphic-unfetch

После завершения процесса установки откройте файл /pages/index.js в редакторе кода.Во-первых, убедитесь, что вы импортировали только что установленный пакет NPM с изоморфной выборкой в верхней части файла:

import fetch from "isomorphic-unfetch"

Это сделает пакет NPM доступным для использования на странице с переменной fetch.Поскольку мы хотим сделать запрос к API RandomDog до загрузки страницы в браузере, нам нужно создать функцию getInitialProps () в верхней части компонента React страницы.Вот как выглядит эта функция:

staticasyncgetInitialProps() {

constres = await

fetch("https://random.dog/woof.json?filter=mp4,webm")

const data = await res.json()

return {

imageURL: data.url

}

}

ФункцияgetInitialProps () являетсяасинхронной, чтоозначает, чтокомпонентReactнебудетотображаться, покаоннезавершитработуиневернетсвоизначениячерезобъектreturn {}.ВнутрифункцииgetInitialProps () мысначалаиспользуемfetch, чтобысделатьHTTP-запроскAPIRandomDogпоURL-адресу<https://random.dog/woof.json?filter=mp4,webm>. Этот HTTP-запрос вернет данные JSON, которые содержат размер файла и URL-адрес случайно сгенерированного изображения:

{"fileSizeBytes": 79732, "url": "https://random.dog/474feef3-7ccf-4aff-94e5-9ad5a8b65a48.jpg"}

Когда запрос завершается внутри функции getInitialProps (), мы сохраняем ответ в переменной данных с помощью метода await res.json().Затем мы возвращаем значение imageURL из функции, которое будет доступно на странице как this.props.imageURL.Теперь, когда наш код настроен на получение случайного изображения собаки при каждой загрузке страницы, нам нужно обновить компонент React для отображения изображения внутри функции render:

render () {

return (

<div className="homepage-wrapper">

<h1>Random Dog Image</h1>

<img src={this.props.imageURL} />

</div>

)

}

Внутри функции render () мы добавили базовый HTML-код для элемента <h1> и <img />, заполненный значением this.props.imageURL, полученным из API RandomDog. Для справки, весь файл /pages/index.js должен выглядеть так:

import React, { Component } from "react"

import fetch from "isomorphic-unfetch"

export default class extends Component {

static async getInitialProps() {

const res = await

fetch("https://random.dog/woof.json?filter=mp4,webm")

const data = await res.json()

return {

imageURL: data.url

}

}

render () {

return (

<div className="homepage-wrapper">

<h1>Random Dog Image</h1>

<img src={this.props.imageURL} />

</div>

)

}

}

Сохраните изменения, внесенные в файл, и просмотрите страницу в своем браузере: http: // localhost: 3000.

В React есть две функции, которые позволяют очень легко применить разделение кода и отложенную загрузку к компонентам React: React.lazy () и React.Suspense.

React.lazy () — это функция, которая позволяет отображать динамический импорт как обычный компонент. Динамический импорт — это способ разделения кода, который играет центральную роль в отложенной загрузке. Основная функция React 16.6, React.lazy (), устраняет необходимость использования сторонней библиотеки, такой как react-loadable.

React.Suspense позволяет указать индикатор загрузки в случае, если компоненты в дереве под ним еще не готовы к рендерингу.

Прежде чем мы увидим React.lazy и React.Suspense в действии, давайте быстро рассмотрим концепции разделения кода и динамического импорта, объясним, как они работают, и разберем, как они облегчают отложенную загрузку в React.

С появлением модулей ES, транспиляторов, таких как Babel, и сборщиков, таких как webpack и Browserify, теперь вы можете писать приложения JavaScript в полностью модульной структуре для облегчения сопровождения. Обычно каждый модуль импортируется и объединяется в один файл, называемый пакетом, затем этот пакет включается на веб-страницу для загрузки всего приложения. По мере роста приложения размер пакета увеличивается и в конечном итоге влияет на время загрузки страницы.

Разделение кода — это процесс разделения большого пакета кода на несколько пакетов, которые могут быть загружены динамически. Это помогает избежать проблем с производительностью, связанных с негабаритными пакетами, без фактического уменьшения объема кода в вашем приложении.

Один из способов разделить код - использовать динамический импорт, который использует синтаксис import (). Вызов import () для загрузки модуля зависит от обещаний JavaScript. Следовательно, он возвращает обещание, которое выполняется с загруженным модулем или отклоняется, если модуль не может быть загружен.Вот как выглядит динамический импорт вспомогательного модуля для работы с датами для приложения, связанного с webpack:

import ("moment").then(({default:moment}) => {

const tomorrow = moment().startOf("day").add(1,"day");

retirn tomorrow.format("LLL");

}).catch(error => console.error("ошибка во время загрузки модуля"))

Вы можете разместить границу ошибки в любом месте над ленивым компонентом, чтобы улучшить взаимодействие с пользователем в случае, если ленивый компонент не загружается. Это очень простая демонстрация на, чтобы показать использование React.lazy () и Suspense для отложенной загрузки компонентов.Вот как выглядит:

import React, { Suspense } from "react";

import Loader from "./components/Loader";

import Header from "./components/Header";

import ErrorBoundary from "./components/ErrorBoundary";

const Calendar = React.lazy(() => {

return new Promise(resolve => setTimeout(resolve, 5 \* 1000)).then(

() =>

Math.floor(Math.random() \* 10) >= 4

? import("./components/Calendar")

: Promise.reject(new Error())

);

});

export default function CalendarComponent() {

return (

<div>

<ErrorBoundary>

<Header>Calendar</Header>

<Suspense fallback={<Loader />}>

<Calendar />

</Suspense>

</ErrorBoundary>

</div>

);

}

React.lazy () и React.Suspense позволяют выполнять разделение кода на основе маршрута без использования внешнего пакета. Вы можете просто преобразовать компоненты маршрута вашего приложения в ленивые компоненты и обернуть все маршруты компонентом Suspense.В следующем фрагменте кода показано разделение кода на основе маршрута с использованием библиотеки Reach Router.

import React, { Suspense } from 'react';

import { Router } from '@reach/router';

import Loading from './Loading';

const Home = React.lazy(() => import('./Home'));

const Dashboard = React.lazy(() => import('./Dashboard'));

const Overview = React.lazy(() => import('./Overview'));

const History = React.lazy(() => import('./History'));

const NotFound = React.lazy(() => import('./NotFound'));

function App() {

return (

<div>

<Suspense fallback={<Loading />}>

<Router>

<Home path="/" />

<Dashboard path="dashboard">

<Overview path="/" />

<History path="/history" />

</Dashboard>

<NotFound default />

</Router>

</Suspense>

</div>

)

}

## **2.2. Разработка чат виджета на основе Socket.IO**

Для начала создайте файл с именем app.js в каталоге вашего проекта. Этобудетсодержатьфактическийсервер:

const express = require("express");

const http = require("http");

const socketIo = require("socket.io");

const port = process.env.PORT || 4001;

const index = require("./routes/index");

const app = express();

app.use(index);

const server = http.createServer(app);

const io = socketIo(server); // < Interesting!

const getApiAndEmit = "TODO";

Если вы раньше работали с Node.js, приведенный выше код не должен быть для вас загадкой. Вы можете заметить вызов socketIo () для инициализации нового экземпляра путем передачи объекта сервера.Приложение также вызывает индексный маршрут: даже если сервер не будет обслуживать какой-либо HTML-контент, нам понадобится очень простой маршрут для прослушивания любого входящего соединения.Создайте файл с именем index.js внутри каталога маршрутов:

constexpress = require("express");

const router = express.Router();

router.get("/", (req, res) => {

res.send({ response: "I am alive" }).status(200);

});

module.exports = router;

Первый и самый важный метод, который вы увидите при работе с Socket.IO, - это on (). Он принимает два аргумента: имя события, в данном случае «соединение», и обратный вызов, который будет выполняться после каждого события соединения.

Событие подключения возвращает объект сокета, который будет передан функции обратного вызова. Используя указанный сокет, вы сможете отправлять данные обратно клиенту в режиме реального времени.

Для того чтобы знать текущее время, мы можем использовать setInterval внутри обратного вызова, а внутри setInterval мы можем использовать другую стрелочную функцию для вызова getApiAndEmit.

Мы также должны обращать внимание на очистку интервала при любом новом подключении и при отключении, чтобы избежать переполнения сервера. Вот идея:

let interval;

io.on("connection", (socket) => {

console.log("New client connected");

if (interval) {

clearInterval(interval);

}

interval = setInterval(() => getApiAndEmit(socket), 1000);

socket.on("disconnect", () => {

console.log("Client disconnected");

clearInterval(interval);

});

});

Теперь все еще в index.js мы можем заставить приложение прослушивать входящие соединения:

server.listen(port, () => console.log(`Listening on port ${port}`));

Также реализуем getApiAndEmit. В качестве аргумента он принимает сокет, который представляет собой не что иное, как канал связи между клиентом и сервером:

const getApiAndEmit = socket => {

const response = new Date();

// Emitting a new message. Will be consumed by the client

socket.emit("FromAPI", response);

};

Функция принимает сокет в качестве аргумента и выдает сообщение «FromAPI», которое будет содержать метку времени.Сообщение может быть перехвачено клиентом Socket.IO во внешнем интерфейсе.С точки зрения сервера, мы закончили, и полный код для app.js должен выглядеть так:

const express = require("express");

const http = require("http");

const socketIo = require("socket.io");

const port = process.env.PORT || 4001;

const index = require("./routes/index");

const app = express();

app.use(index);

const server = http.createServer(app);

const io = socketIo(server);

let interval;

io.on("connection", (socket) => {

console.log("New client connected");

if (interval) {

clearInterval(interval);

}

interval = setInterval(() => getApiAndEmit(socket), 1000);

socket.on("disconnect", () => {

console.log("Client disconnected");

clearInterval(interval);

});

});

const getApiAndEmit = socket => {

const response = new Date();

// Emitting a new message. Will be consumed by the client

socket.emit("FromAPI", response);

};

server.listen(port, () => console.log(`Listening on port ${port}`));

Как только сервер запустится, вы увидите следующий вывод: «Прослушивание порта 4001», который подтверждает, что все работает нормально.

Для простоты в качестве шаблона клиентской части чат виджета используем приложение create-react-app. Следует обрадить внимание чо это нужно делать уже вне папки с сервером.

npx create-react-app socket-io-client

Затем нужно перейти внутрь проекта и установить клиент Socket.IO:

npm i socket.io-client

Для простоты мы будем использовать App.js в каталоге src. В файле App.js, удаляется все содержимое заменяется следующим кодом:

import React, { useState, useEffect } from "react";

import socketIOClient from "socket.io-client";

const ENDPOINT = "http://127.0.0.1:4001";

function App() {

const [response, setResponse] = useState("");

useEffect(() => {

const socket = socketIOClient(ENDPOINT);

socket.on("FromAPI", data => {

setResponse(data);

});

}, []);

return (

<p>

It's <time dateTime={response}>{response}</time>

</p>

);

}

export default App;

Вэтомпримеремыиспользуем React Hooks, аименно useEffect и useState. Если вы не знаете, что такое хук, ознакомьтесь с руководством по React Hooks.Основная теория здесь заключается в том, что мы подключаемся к серверу сокетов при монтировании компонента с помощью useEffect. Затем мы сохраняем каждое новое входящее сообщение в состоянии компонента.В этой логике нам нужно исправить еще одну деталь, а пока сохраните и закройте файл, откройте терминал, перейдите в папку сервера и запустите сервер Socket.io:

d socket-io-server && node app.js

В другом терминале перемещаясь в папку клиента запускается проект React:

cd socket-io-client && npm start

Следует помнить, сокет - это канал связи, и мы можем отслеживать любые события, происходящие внутри него.Клиент может прослушивать событие с помощью метода on (). После открытия соединения интерфейс может получать обновления, не обновляя страницу.

Часто упускается из виду, что логика очистки таймеров и слушателей в JavaScript действительно имеет первостепенное значение для предотвращения утечек памяти во внешнем интерфейсе.Рассмотрим следующие примеры, в которых мы перемещаем клиент Socket.Io в автономный компонент:

import React, { useEffect, useState } from "react";

import socketIOClient from "socket.io-client";

const ENDPOINT = "http://127.0.0.1:4001";

export default function ClientComponent() {

const [response, setResponse] = useState("");

useEffect(() => {

const socket = socketIOClient(ENDPOINT);

socket.on("FromAPI", data => {

setResponse(data);

});

}, []);

return (

<p>

It's <time dateTime={response}>{response}</time>

</p>

);

}

Затем в компонент верхнего уровня мы включаем клиента и некоторую логику для его удаления из DOM:

import React, { useState } from "react";

import ClientComponent from "./ClientComponent";

function App() {

const [loadClient, setLoadClient] = useState(true);

return (

<>

{/\* LOAD OR UNLOAD THE CLIENT \*/}

<button onClick={() => setLoadClient(prevState => !prevState)}>

STOP CLIENT

</button>

{/\* SOCKET IO CLIENT\*/}

{loadClient ?<ClientComponent /> : null}

</>

);

}

export default App;

Здесь происходит то, что мы переключаем внешний вид ClientComponent, нажимая кнопку. Однако есть то, что может пойти не так, и вся суть в деталях, когда ClientComponent отключается от DOM, соединение Socket.Io остается активным. Ещеразобратимвниманиенахук:

useEffect(() => {

const socket = socketIOClient(ENDPOINT);

socket.on("FromAPI", data => {

setResponse(data);

});

}, []);

Перехватчик useEffect здесь имеет массив пустых зависимостей, который гарантирует, что перехватчик запускается только один раз.Это нормально, но нам также нужно закрыть соединение, когда компонент исчезнет из DOM.Для этого мы возвращаем функцию из useEffect с вызовом функции disconnect () на клиенте:

useEffect(() => {

const socket = socketIOClient(ENDPOINT);

socket.on("FromAPI", data => {

setResponse(data);

});

// CLEAN UP THE EFFECT

return () => socket.disconnect();

//

}, []);

Таким образом, эффективно закрывается соединение при отключении компонента.

# 3. ФУНКЦИОНАЛ САЙТА

## **3.1 Виджет чата**

Пользователи могут обратится за помощью приватным способом – задать вопрос через виджет. Для получения обратной связи им необходимо указать контактные данные. Администраторы сайта, в качестве которых могут выступать квалифицированные специалисты, смогут прочитать обращение в админской панели и отреагировать. На сайте виджет выглядит так:

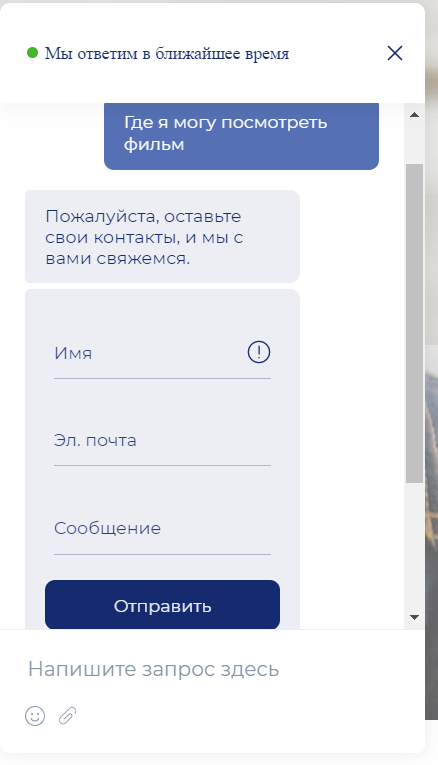


Рисунок 5 – Виджет чата

Упрощенный вариант веб-сокета на стороне сервера:

var express = require('express');

var app = express();

var server = require('http').createServer(app);

var io = require('socket.io')(server);

var redis = require('socket.io-redis');

var port = process.env.PORT || 3000;

var serverName = process.env.NAME || 'Unknown';

io.adapter(redis({ host: 'redis', port: 6379 }));

server.listen(port, function () {

console.log('Server listening at port %d', port);

console.log('Hello, I\'m %s, how can I help?', serverName);

});

// Routing

app.use(express.static(\_\_dirname + '/public'));

// Chatroom

var numUsers = 0;

io.on('connection', function (socket) {

socket.emit('my-name-is', serverName);

var addedUser = false;

// when the client emits 'new message', this listens and executes

socket.on('new message', function (data) {

// we tell the client to execute 'new message'

socket.broadcast.emit('new message', {

username: socket.username,

message: data

});

});

// when the client emits 'add user', this listens and executes

socket.on('add user', function (username) {

if (addedUser) return;

// we store the username in the socket session for this client

socket.username = username;

++numUsers;

addedUser = true;

socket.emit('login', {

numUsers: numUsers

});

// echo globally (all clients) that a person has connected

socket.broadcast.emit('user joined', {

username: socket.username,

numUsers: numUsers

});

});

// when the client emits 'typing', we broadcast it to others

socket.on('typing', function () {

socket.broadcast.emit('typing', {

username: socket.username

});

});

// when the client emits 'stop typing', we broadcast it to others

socket.on('stop typing', function () {

socket.broadcast.emit('stop typing', {

username: socket.username

});

});

// when the user disconnects.. perform this

socket.on('disconnect', function () {

if (addedUser) {

--numUsers;

// echo globally that this client has left

socket.broadcast.emit('user left', {

username: socket.username,

numUsers: numUsers

});

}

});

});

Для хранения пользователей, их коментариев и многого другого используется не реаляционная база данные MongoDB. Упращенный вариант кода подключения и элементарного взаимодействия ниже:

require('dotenv').config();

constfs = require('fs');

const join = require('path').join;

const express = require('express');

const mongoose = require('mongoose');

const passport = require('passport');

const config = require('./config');

const models = join(\_\_dirname, 'app/models');

const port = process.env.PORT || 3000;

const app = express();

/\*\*

\* Expose

\*/

module.exports = app;

// Bootstrap models

fs.readdirSync(models)

.filter(file => ~file.search(/^[^.].\*\.js$/))

.forEach(file => require(join(models, file)));

// Bootstrap routes

require('./config/passport')(passport);

require('./config/express')(app, passport);

require('./config/routes')(app, passport);

connect();

function listen() {

if (app.get('env') === 'test') return;

app.listen(port);

console.log('Express app started on port ' + port);

}

function connect() {

mongoose.connection

.on('error', console.log)

.on('disconnected', connect)

.once('open', listen);

return mongoose.connect(config.db, {

keepAlive: 1,

useNewUrlParser: true,

useUnifiedTopology: true

});

}

## **3.2Форум с оценками и возможностью менторства**

Реализованная возможность коментировать, оставлять оценку, а так же присоединяться к качестве наблюдателя выглядит так:

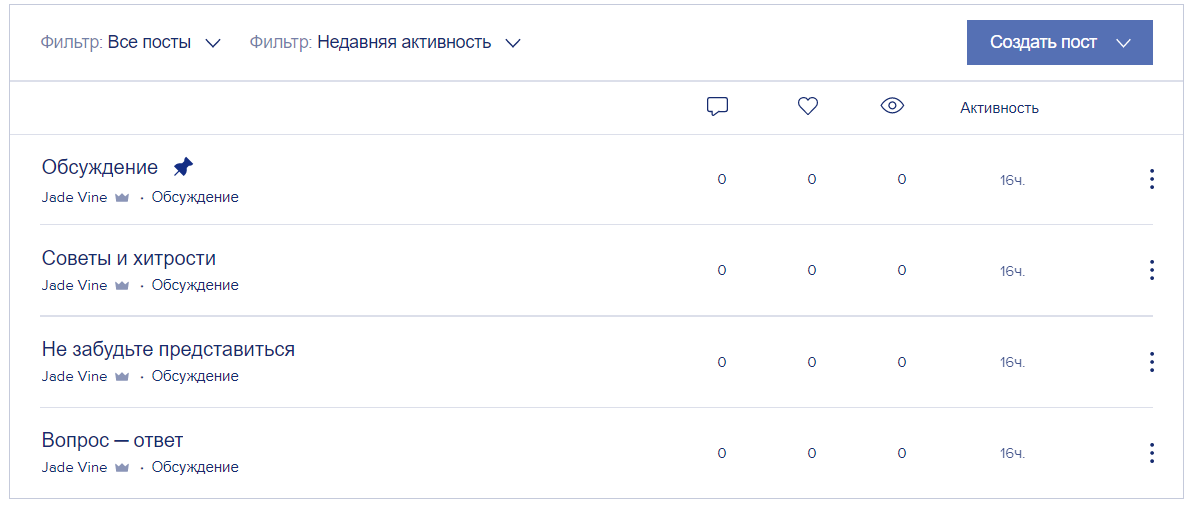


Рисунок 6 – Форум

Исходны код одного из компонентов списка можно наблюдать ниже:

import { Select, Label, Spinner, Row, Column, Icon } from '@hospitalrun/components'

import React, { useEffect, ReactElement } from 'react'

import { SelectOption } from '../shared/components/input/SelectOption'

import TextFieldWithLabelFormGroup from '../shared/components/input/TextFieldWithLabelFormGroup'

import TextInputWithLabelFormGroup from '../shared/components/input/TextInputWithLabelFormGroup'

import useTranslator from '../shared/hooks/useTranslator'

import { ContactInfoPiece } from '../shared/model/ContactInformation'

import { uuid } from '../shared/util/uuid'

import ContactInfoTypes from './ContactInfoTypes'

interface Props {

component: 'TextInputWithLabelFormGroup' | 'TextFieldWithLabelFormGroup'

data: ContactInfoPiece[]

errors?: (string | undefined)[]

label: string

name: string

isEditable?: boolean

onChange?: (newData: ContactInfoPiece[]) => void

}

const ContactInfo = (props: Props): ReactElement => {

const { component, data, errors, label, name, isEditable, onChange } = props

const { t } = useTranslator()

useEffect(() => {

if (onChange && data.length === 0) {

onChange([...data, { id: uuid(), value: '' }])

}

}, [data, onChange])

const typeOptions: SelectOption[] = Object.values(ContactInfoTypes).map((value) => ({

label: t(`patient.contactInfoType.options.${value}`),

value: `${value}`,

}))

const header = (

<Row className="header mb-2">

<Column xs={12} sm={4}>

<span className="">{t('patient.contactInfoType.label')}</span>

<span className="d-sm-none">&amp; {t(label)}</span>

</Column>

<Column className="d-none d-sm-block" sm={8}>

{t(label)}

</Column>

</Row>

)

const componentList = {

TextInputWithLabelFormGroup,

TextFieldWithLabelFormGroup,

}

const Component = componentList[component]

const onTypeChange = (newType: string, index: number) => {

if (onChange) {

const currentContact = { ...data[index], type: newType }

const newContacts = [...data]

newContacts.splice(index, 1, currentContact)

onChange(newContacts)

}

}

const onValueChange = (

event: React.ChangeEvent<HTMLInputElement | HTMLTextAreaElement>,

index: number,

) => {

if (onChange) {

const newValue = event.currentTarget.value

const currentContact = { ...data[index], value: newValue }

const newContacts = [...data]

newContacts.splice(index, 1, currentContact)

onChange(newContacts)

}

}

const entries = data.map((entry, i) => {

const error = errors ? errors[i] : undefined

return (

<Row key={entry.id}>

<Column sm={4}>

<div className="form-group" data-testid={`${name}Type${i}Select`}>

<Label text={`${name}Type${i}`} />

<Select

id={`${name}Type${i}Select`}

options={typeOptions}

defaultSelected={typeOptions.filter(({ value }) => value === entry.type)}

onChange={(values) => onTypeChange(values[0], i)}

disabled={!isEditable}

/>

</div>

</Column>

<Column sm={8}>

<Component

name={`${name}${i}`}

value={entry.value}

isEditable={isEditable}

onChange={(event: any) => onValueChange(event, i)}

feedback={error && t(error)}

isInvalid={!!error}

/>

</Column>

</Row>

)

})

const onAddClick = () => {

if (!onChange) {

return

}

// 1. pick up only non-empty string

const newData = data.filter(({ value }) => value.trim() !== '')

// 2. add a new entry

newData.push({ id: uuid(), value: '' })

// 3. send updates

onChange(newData)

}

const addButton = (

<div className="text-right">

<button type="button" className="btn btn-link" onClick={onAddClick}>

<Icon icon="add" /> {t('actions.add')}

</button>

</div>

)

if (isEditable && data.length === 0) {

return (

<Spinner

aria-hidden="false"

aria-label="Loading"

color="blue"

loading

size={20}

type="SyncLoader"

/>

)

}

return (

<div>

{data.length> 0 ? header : null}

{entries}

{isEditable ? addButton : null}

</div>

)

}

ContactInfo.defaultProps = {

data: [],

}

export default ContactInfo

subordinate\_dept = Employee.objects.filter(Q(owner=user\_id) & Q(confirmed\_head=True)).values\_list('dept')

context['unconfirmed\_print\_count'] = Employee.objects.filter (dept\_\_in=subordinate\_dept,confirmed\_print=False).select\_related().count()

if is\_manager:

context['unconfirmed\_management\_count']=Employee.objects.filter((Q(post='заведующийкафедрой')|Q(post='заведующийлабораторией'))&Q(confirmed\_head=False)).select\_related().count()

context['is\_manager'] = is\_manager

context['is\_head'] = is\_head

context['is\_print'] = is\_print

context['employee\_list'] = employee\_list

Если пользоваль желает выступать наблюдателем, его данные отправляются на сервер, и все участники данного поста на форуме имеют возмодность посмотреть его профиль.

## **3.3 Панель администратора для контроля контента управления сайтом**

Панель администраторатоже создана с помощью Next.jsиReact, часть исходного кода панели:

import React from "react";

import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';

import { faAngleDown, faAngleUp, faChartArea, faChartBar, faChartLine, faFlagUsa, faFolderOpen, faGlobeEurope, faPaperclip, faUserPlus } from '@fortawesome/free-solid-svg-icons';

import { faAngular, faBootstrap, faReact, faVuejs } from "@fortawesome/free-brands-svg-icons";

import { Col, Row, Card, Image, Button, ListGroup, ProgressBar } from '@themesberg/react-bootstrap';

import { CircleChart, BarChart, SalesValueChart, SalesValueChartphone } from "./Charts";

import Profile1 from "../assets/img/team/profile-picture-1.jpg";

import ProfileCover from "../assets/img/profile-cover.jpg";

import teamMembers from "../data/teamMembers";

export const ProfileCardWidget = () => {

return (

<Card border="light" className="text-center p-0 mb-4">

<div style={{ backgroundImage: `url(${ProfileCover})` }} className="profile-cover rounded-top" />

<Card.Body className="pb-5">

<Card.Img src={Profile1} alt="Neil Portrait" className="user-avatar large-avatar rounded-circle mx-auto mt-n7 mb-4" />

<Card.Title>Neil Sims</Card.Title>

<Card.Subtitle className="fw-normal">Senior Software Engineer</Card.Subtitle>

<Card.Text className="text-gray mb-4">New York, USA</Card.Text>

<Button variant="primary" size="sm" className="me-2">

<FontAwesomeIcon icon={faUserPlus} className="me-1" /> Connect

</Button>

<Button variant="secondary" size="sm">Send Message</Button>

</Card.Body>

</Card>

);

};

export const ChoosePhotoWidget = (props) => {

const { title, photo } = props;

return (

<Card border="light" className="bg-white shadow-sm mb-4">

<Card.Body>

<h5 className="mb-4">{title}</h5>

<div className="d-xl-flex align-items-center">

<div className="user-avatar xl-avatar">

<Image fluid rounded src={photo} />

</div>

<div className="file-field">

<div className="d-flex justify-content-xl-center ms-xl-3">

<div className="d-flex">

<span className="icon icon-md">

<FontAwesomeIcon icon={faPaperclip} className="me-3" />

</span>

<input type="file" />

<div className="d-md-block text-start">

<div className="fw-normal text-dark mb-1">Choose Image</div>

<div className="text-gray small">JPG, GIF or PNG. Max size of 800K</div>

</div>

</div>

</div>

</div>

</div>

</Card.Body>

</Card>

);

};

export default (props) => {

const [notifications, setNotifications] = useState(NOTIFICATIONS\_DATA);

const areNotificationsRead = notifications.reduce((acc, notif) => acc && notif.read, true);

const markNotificationsAsRead = () => {

setTimeout(() => {

setNotifications(notifications.map(n => ({ ...n, read: true })));

}, 300);

};

const Notification = (props) => {

const { link, sender, image, time, message, read = false } = props;

const readClassName = read ? "" : "text-danger";

return (

<ListGroup.Item action href={link} className="border-bottom border-light">

<Row className="align-items-center">

<Col className="col-auto">

<Image src={image} className="user-avatar lg-avatar rounded-circle" />

</Col>

<Col className="ps-0 ms--2">

<div className="d-flex justify-content-between align-items-center">

<div>

<h4 className="h6 mb-0 text-small">{sender}</h4>

</div>

<div className="text-end">

<small className={readClassName}>{time}</small>

</div>

</div>

<p className="font-small mt-1 mb-0">{message}</p>

</Col>

</Row>

</ListGroup.Item>

);

};

return (

<Navbar variant="dark" expanded className="ps-0 pe-2 pb-0">

<Container fluid className="px-0">

<div className="d-flex justify-content-between w-100">

<div className="d-flex align-items-center">

<Form className="navbar-search">

<Form.Group id="topbarSearch">

<InputGroup className="input-group-merge search-bar">

<InputGroup.Text><FontAwesomeIcon icon={faSearch} /></InputGroup.Text>

<Form.Control type="text" placeholder="Search" />

</InputGroup>

</Form.Group>

</Form>

</div>

<Nav className="align-items-center">

<Dropdown as={Nav.Item} onToggle={markNotificationsAsRead} >

<Dropdown.Toggle as={Nav.Link} className="text-dark icon-notifications me-lg-3">

<span className="icon icon-sm">

<FontAwesomeIcon icon={faBell} className="bell-shake" />

{areNotificationsRead ? null : <span className="icon-badge rounded-circle unread-notifications" />}

</span>

</Dropdown.Toggle>

<Dropdown.Menu className="dashboard-dropdown notifications-dropdown dropdown-menu-lg dropdown-menu-center mt-2 py-0">

<ListGroup className="list-group-flush">

<Nav.Link href="#" className="text-center text-primary fw-bold border-bottom border-light py-3">

Notifications

</Nav.Link>

{notifications.map(n =><Notification key={`notification-${n.id}`} {...n} />)}

<Dropdown.Item className="text-center text-primary fw-bold py-3">

View all

</Dropdown.Item>

</ListGroup>

</Dropdown.Menu>

</Dropdown>

<Dropdown as={Nav.Item}>

<Dropdown.Toggle as={Nav.Link} className="pt-1 px-0">

<div className="media d-flex align-items-center">

<Image src={Profile3} className="user-avatar md-avatar rounded-circle" />

<div className="media-body ms-2 text-dark align-items-center d-none d-lg-block">

<span className="mb-0 font-small fw-bold">Bonnie Green</span>

</div>

</div>

</Dropdown.Toggle>

<Dropdown.Menu className="user-dropdown dropdown-menu-right mt-2">

<Dropdown.Item className="fw-bold">

<FontAwesomeIcon icon={faUserCircle} className="me-2" /> My Profile

</Dropdown.Item>

<Dropdown.Item className="fw-bold">

<FontAwesomeIcon icon={faCog} className="me-2" /> Settings

</Dropdown.Item>

<Dropdown.Item className="fw-bold">

<FontAwesomeIcon icon={faEnvelopeOpen} className="me-2" /> Messages

</Dropdown.Item>

<Dropdown.Item className="fw-bold">

<FontAwesomeIcon icon={faUserShield} className="me-2" /> Support

</Dropdown.Item>

<Dropdown.Divider />

<Dropdown.Item className="fw-bold">

<FontAwesomeIcon icon={faSignOutAlt} className="text-danger me-2" /> Logout

</Dropdown.Item>

</Dropdown.Menu>

</Dropdown>

</Nav>

</div>

</Container>

</Navbar>

);

};

export const CounterWidget = (props) => {

const { icon, iconColor, category, title, period, percentage } = props;

const percentageIcon = percentage < 0 ? faAngleDown : faAngleUp;

const percentageColor = percentage < 0 ? "text-danger" : "text-success";

return (

<Card border="light" className="shadow-sm">

<Card.Body>

<Row className="d-block d-xl-flex align-items-center">

<Col xl={5} className="text-xl-center d-flex align-items-center justify-content-xl-center mb-3 mb-xl-0">

<div className={`icon icon-shape icon-md icon-${iconColor} rounded me-4 me-sm-0`}>

<FontAwesomeIcon icon={icon} />

</div>

<div className="d-sm-none">

<h5>{category}</h5>

<h3 className="mb-1">{title}</h3>

</div>

</Col>

<Col xs={12} xl={7} className="px-xl-0">

<div className="d-none d-sm-block">

<h5>{category}</h5>

<h3 className="mb-1">{title}</h3>

</div>

<small>{period}, <FontAwesomeIcon icon={faGlobeEurope} size="xs" /> WorldWide</small>

<div className="small mt-2">

<FontAwesomeIcon icon={percentageIcon} className={`${percentageColor} me-1`} />

<span className={`${percentageColor} fw-bold`}>

{percentage}%

</span> Since last month

</div>

</Col>

</Row>

</Card.Body>

</Card>

);

};

export const CircleChartWidget = (props) => {

const { title, data = [] } = props;

const series = data.map(d => d.value);

return (

<Card border="light" className="shadow-sm">

<Card.Body>

<Row className="d-block d-xl-flex align-items-center">

<Col xs={12} xl={5} className="text-xl-center d-flex align-items-center justify-content-xl-center mb-3 mb-xl-0">

<CircleChart series={series} />

</Col>

<Col xs={12} xl={7} className="px-xl-0">

<h5 className="mb-3">{title}</h5>

{data.map(d => (

<h6 key={`circle-element-${d.id}`} className="fw-normal text-gray">

<FontAwesomeIcon icon={d.icon} className={`icon icon-xs text-${d.color} w-20 me-1`} />

{` ${d.label} `}{`${d.value}%`}

</h6>

))}

</Col>

# ЗАКЛЮЧЕНИЕ

Для разработки сайта был использован фреймворк Next.Js, который упростил процесс. Он позволяет использовать популярные и востребованные инструменты и технологии, такие как React, SSR.

Были получены следующиерезультаты:

1. Изучены особенности работы с веб-сокетами, современные паттерны проектирования веб-приложений, а также фреймворк Next.js.

2. Практическая часть выпускной квалификационной работы представлена разработкой электронного сервиса по социально-психолого-педагогическому сопровождению семей с детьми раннего возраста. Был разработан сайт с использованиемNext.js, React, WebSockets.

3. Разработанный сайт преследует цель наполнения образовательным контентом для семей с детьми раннего возраста. Он должен регулярно обновляться специалистами социально-психолого-педагогического профиля.

Разработанное программное обеспечение планируется использовать на базе федеральной инновационной площадки «Цифровой информационно-образовательный ресурс сопровождения семей с детьми раннего возраста «Шаг навстречу» Орловского государственного университета имени И.С. Тургенева.

# СПИСОК ИСТОЧНИКОВ

1. Документация по Next.js [Электронныйресурс]. – Режимдоступа: https://nextjs.org/docs.
2. Документация по React.js [Электронный ресурс]. – Режим доступа: https://ru.reactjs.org/docs/getting-started.html.
3. Документация по Nginx [Электронный ресурс]. – Режим доступа: https://nginx.org/ru/docs/.
4. Документация по WebSockets [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/API/WebSockets\_API.
5. Документация по css [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/CSS.
6. Документация по js [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/JavaScript.
7. Документация по Docker [Электронный ресурс]. – Режим доступа: https://docs.docker.com/.